

## Git in 4 Weeks - Part 4 Labs

Revision 1.1 - 06/21/21

Brent Laster for Tech Skill Transformations LLC

### Lab 11 - Working with Worktrees

**Purpose:** In this lab, we'll get some experience with how to work on multiple branches at the same time.

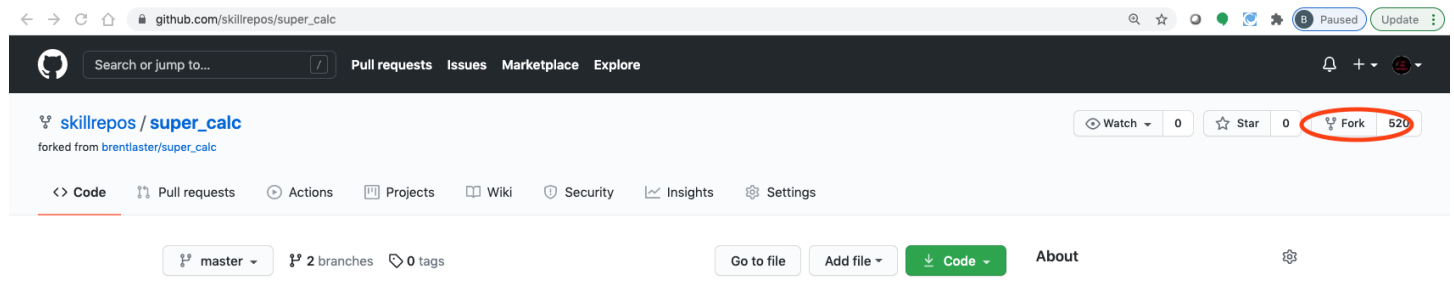
1. In my GitHub space, I have a project called *calc2* which is a simple javascript calculator program. It has multiple branches for *features*, *documentation*, etc. For this lab, I have split it up into three separate projects:
  - *super\_calc*, a version of the *calc2* project with only the master and feature branches
  - *sub\_ui*, a separate repository consisting of only the content of the *ui* branch split out from the *calc2* project
  - *sub\_docs*, a separate repository consisting of only the content of the *docs* branch split out from the *calc2* project

Log in to your GitHub account and fork the three projects from the following listed locations. (As a reminder, the **Fork** button is in the upper-right corner of the pages.) This will prepare your area on GitHub for doing this lab, as well as the labs on subtrees.

[https://github.com/skillrepos/super\\_calc.git](https://github.com/skillrepos/super_calc.git)

[https://github.com/skillrepos/sub\\_ui.git](https://github.com/skillrepos/sub_ui.git)

[https://github.com/skillrepos/sub\\_docs.git](https://github.com/skillrepos/sub_docs.git)



2. In a new directory, clone down the *super\_calc* project that you forked in step 1, using the following command:

```
$ git clone https://github.com/<your github userid>/super_calc.git
```

3. Now, change into the cloned directory - *super\_calc*.

```
$ cd super_calc
```

4. In this case, you want to work on both the master branch and the features branch at the same time. You can work

on the master branch in this directory, but you need to create a separate working tree (worktree) for working on the features branch. You can do that with the worktree add command, passing the -b to create a new local branch from the remote tracking branch.

```
$ git worktree add -b features ../super_calc_features origin/features
```

5. Change into the new subdirectory with the new worktree. Note that you are on the features branch. Edit the calc.html file and update the line in the file surrounded by <title> and </title>. The process is described below.

```
$ cd ../super_calc_features
```

```
$ git branch (note which one is selected automatically)
```

**Edit calc.html and change**

```
<title>Calc</title>
```

**to**

```
<title> github_user_id's Calc</title>
```

***substituting in your GitHub user ID for "github\_user\_id".***

6. Save your changes and commit them back into the repository.

```
$ git commit -am "Updating title"
```

7. Switch over to your original worktree.

```
$ cd ../super_calc
```

8. Look at what branches you have there.

```
$ git branch
```

9. Note that you have the features branch you created for the other worktree. Do a log on that branch; you can see your new commit just as if you had done it in this worktree.

```
$ git log --oneline features
```

10. You no longer need your separate worktree. However, before you remove it, take a look at what worktrees are

currently there.

```
$ git worktree list
```

11. You can now remove the worktree. First, remove the actual directory; then use the prune option to get rid of the worktree reference.

```
$ rm -rf ../super_calc_features
$ git worktree prune
```

```
=====
                        END OF LAB
=====
```

## Lab 12 - Working with Pull Requests

**Purpose:** In this lab, we'll see how to create and merge Pull Requests within a GitHub repository

In the last lab, you created a new branch named "features" and committed a change into it. We can leverage this commit to create a Pull Request to work with in GitHub.

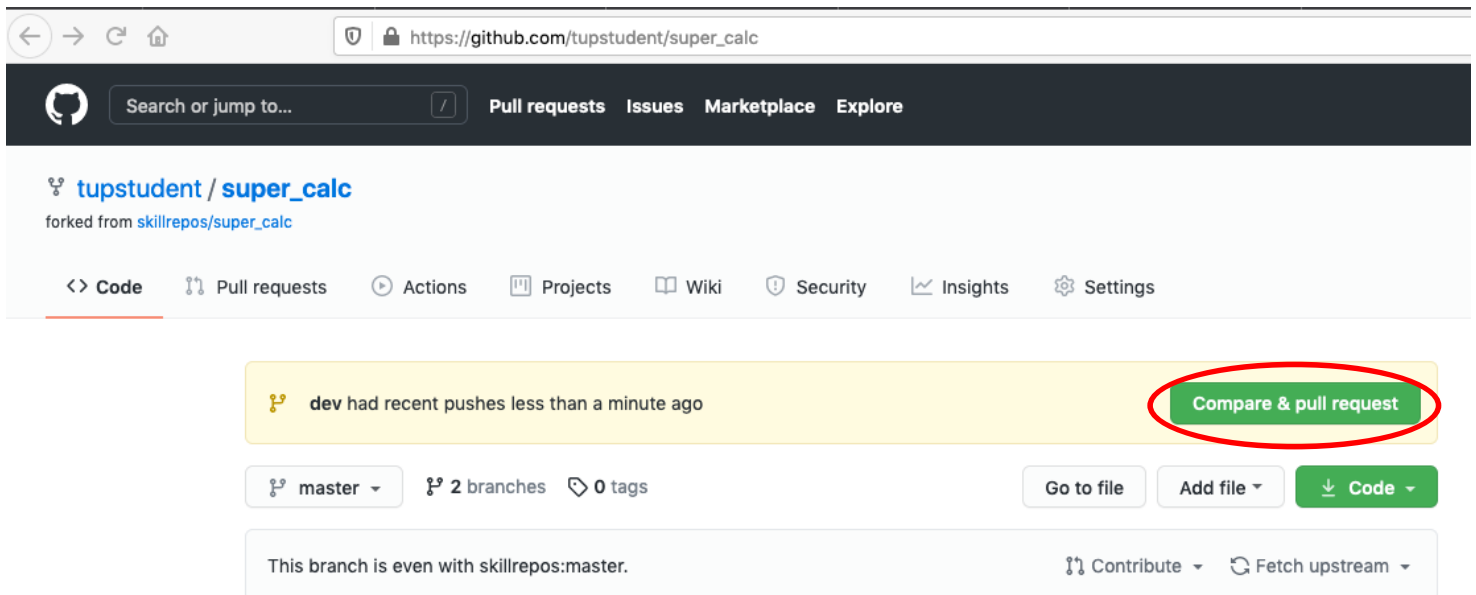
1. Let's go ahead and push the change over to our remote on the GitHub side to a new branch. (You can use a different name in place of "dev" if you want. Just use that name consistently wherever "dev" is used here.)

```
$ git push origin features:dev
```

2. In addition to pushing your changes to the new branch, GitHub will also provide a link in the output that offers you you should see that GitHub is giving you a link that you can go to create a pull request for this change.

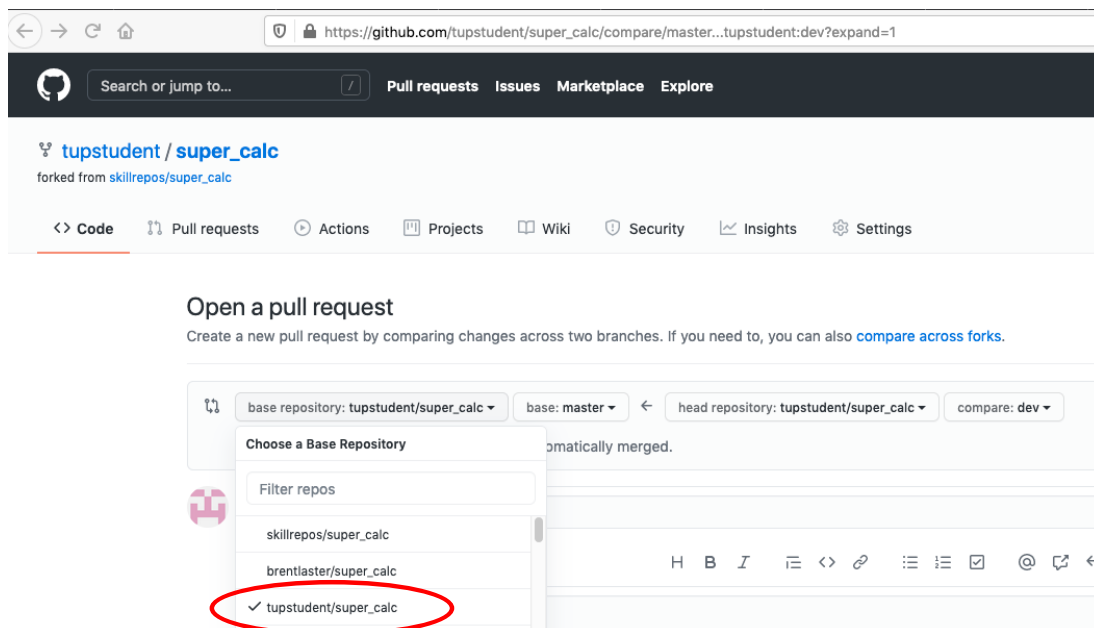
```
:super_calc developer$ git push origin features:dev
Enumerating objects: 5, done.
Counting objects: 100% (5/5), done.
Delta compression using up to 16 threads
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 292 bytes | 292.00 KiB/s, done.
Total 3 (delta 1), reused 0 (delta 0)
remote: Resolving deltas: 100% (1/1), completed with 1 local object.
remote:
remote: Create a pull request for 'dev' on GitHub by visiting:
remote:   https://github.com/tupstudent/super_calc/pull/new/dev
remote:
To https://github.com/tupstudent/super_calc
 * [new branch]      features -> dev
```

3. If you go back to the repository location now in GitHub (github.com/<your github id>/super\_calc), you should see an option now to do a "Compare & (create a) pull request" via a large green button. Go ahead and click that.



4. You'll now be on the screen to open a pull request. (This is the same screen you would get to if you followed the link in the push output in step 2.)

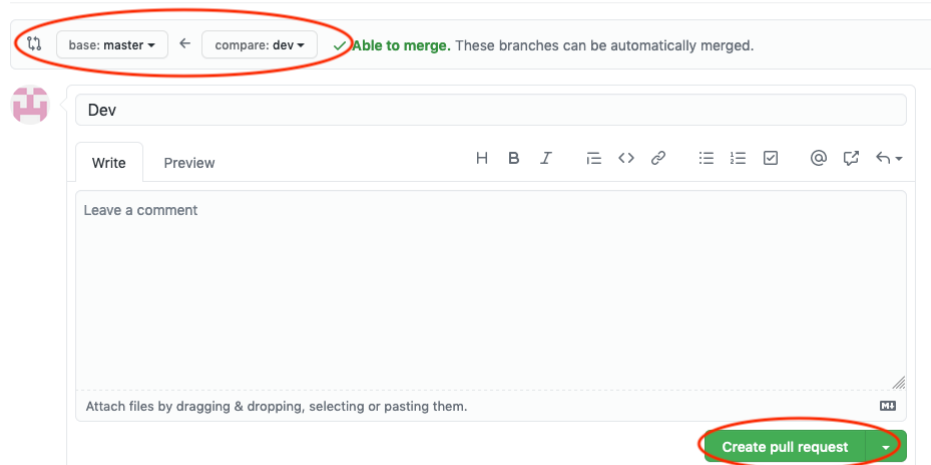
You can update the title or text below it if you want. In the top row (the one with "Able to merge"), change the left dropdown to first have the same project as the right dropdown (both with your GitHub project).



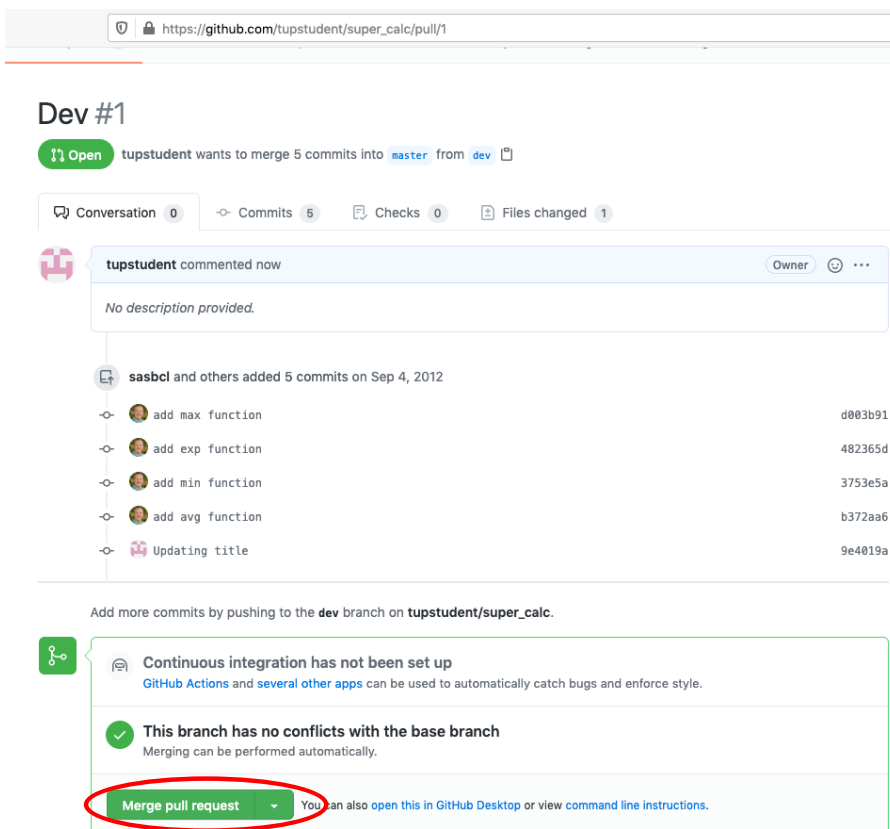
- Then select the "master" branch in the left and the "dev" branch in the right dropdown - as shown in the figure. Afterwards, you can click on the "Create pull request" button.

### Open a pull request

Create a new pull request by comparing changes across two branches. If you need to, you can also [compare across forks](#).






- The pull request will open at "github.com/<your github userid>/super\_calc/pull/1". You will now be able to see the changes that will be merged in if we proceed. When ready, go ahead and select the "Merge pull request" button.





- Click on the "Confirm merge" button.


## Dev #1


 **Open** tupstudent wants to merge 5 commits into `master` from `dev` 

 Conversation 0

 Commits 5


 Checks 0

 Files changed 1





**tupstudent** commented 3 minutes ago Owner 😊 ...

No description provided.





**sasbcl** and others added 5 commits on Sep 4, 2012




 add max function


d003b91




 add exp function


482365d




 add min function


3753e5a



 add avg function

b372aa6



 Updating title

9e4019a

Add more commits by pushing to the `dev` branch on **tupstudent/super\_calc**.



Merge pull request #1 from tupstudent/dev



Dev





**Confirm merge**


Cancel

- After this, you should see the screen indicating your Merge Request has been successfully merged and closed.


# Dev #1



 **Merged** tupstudent merged 5 commits into `master` from `dev`  now



 Conversation **0**  Commits **5**  Checks **0**  Files changed **1**



 **tupstudent** commented 5 minutes ago Owner 😊 ...



No description provided.



 **sasbcl** and others added 5 commits on Sep 4, 2012



  add max function d003b91

  add exp function 482365d

  add min function 3753e5a

  add avg function b372aa6

  Updating title 9e4019a

  **tupstudent** merged commit **c27952c** into `master` now Revert

9. Let's do one more. Back in your terminal, make sure you are on the master branch. Then create a new branch called "test".

```
$ git checkout master
```

```
$ git checkout -b test
```

10. Then, as you did before, edit the calc.html file and add in "Test" instead of your GitHub userid.

**Edit calc.html and change**

```
<title>Calc</title>
```

to

```
<title> Test Calc</title>
```

10. Save your changes and commit them back into the repository. Then push the new branch back to the remote.

```
$ git commit -am "add test title"
```

```
$ git push origin test:test
```

11. You'll see the same automatic message giving you a URL to go to create a new pull request. Something like:

```
Total 3 (delta 1), reused 0 (delta 0)
remote: Resolving deltas: 100% (1/1), completed with 1 local object.
remote:
remote: Create a pull request for 'test' on GitHub by visiting:
remote:   https://github.com/<your github userid>/super\_calc/pull/new/test
remote: To https://github.com/<your github userid>/super\_calc.git
      * [new branch]      test -> test
```

12. Open/go to the link in the middle of the message for "Create a pull request...". You'll be at a similar screen as before. Change the project in the left top dropdown to be the same project as the one on the right. (As you did back in step 4.) At that point, the targets will switch to "base:master" and "compare:test". But there will be a warning message "Can't automatically merge." Go ahead and click the button to "Create pull request".

The screenshot shows the GitHub interface for creating a pull request. At the top, there's a navigation bar with links for Code, Pull requests, Actions, Projects, Wiki, Security, Insights, and Settings. Below this, the heading 'Open a pull request' is followed by a subtext: 'Create a new pull request by comparing changes across two branches. If you need to, you can also [compare across forks](#).' The main area shows a comparison between 'base: master' and 'compare: test'. A red circle highlights these dropdowns. To the right of the dropdowns, a red warning message says 'Can't automatically merge. Don't worry, you can still create the pull request.' Below this, there's a text input field with 'add test title'. Underneath, there are tabs for 'Write' and 'Preview'. The 'Write' tab is active, showing a 'Leave a comment' section. At the bottom right, a green 'Create pull request' button is highlighted with a red circle.

13. Notice the error message near the bottom "This branch has conflicts that must be resolved". Also there is a "Resolve conflicts" button. Click on that.



forked from [skillrepos/super\\_calc](#)

<> Code   Pull requests 1   Actions   Projects   Wiki   Security   Insights   Settings

## add test title #2

Open   techupskills wants to merge 1 commit into `master` from `test`

Conversation 0   Commits 1   Checks 0   Files changed 1

techupskills commented now   Owner   ...

No description provided.

add test title   5c6f1ce

Add more commits by pushing to the `test` branch on [techupskills/super\\_calc](#).

**This branch has conflicts that must be resolved**   **Resolve conflicts**

Use the [web editor](#) or the [command line](#) to resolve conflicts.

**Conflicting files**

calc.html

Merge pull request   You can also [open this in GitHub Desktop](#) or view [command line instructions](#).

14. This will now put you on a screen showing the conflicts.

## add test title #2

Resolving conflicts between `test` and `master` and committing changes → `test`

1 conflicting file	calc.html
calc.html calc.html	<pre> 1  &lt;html&gt; 2  &lt;head&gt; 3  &lt;&lt;&lt;&lt;&lt;&lt;&lt; test 4      &lt;title&gt;Test Calc&lt;/title&gt; 5  ===== 6      &lt;title&gt;TechUpskills Calc&lt;/title&gt; 7  &gt;&gt;&gt;&gt;&gt;&gt;&gt; master 8 9  &lt;script language=javascript type="text/javascript"&gt; 10 11  var plus,minus,divide,multiply,max,pow,min,avg </pre>

15. You can highlight and hit the "Del" key to get rid of lines 3, 5, 6, and 7 to remove the markers and resolve the conflict. Then you can click on the "Mark as resolved" button.

## add test title #2

Resolving conflicts between `test` and `master` and committing changes → `test`

1 conflicting file

calc.html

```
1 <html>
2 <head>
3
4   <title>Test Calc</title>
5
6 </head>
7 <script language=javascript type="text/javascript">
8   var plus,minus,divide,multiply,max,pow,min,avg
9
```

1 conflict Prev Next Mark as resolved

16. At this point, the check should turn green, the status should indicate "Resolved" and you can click the new green "Commit merge" button.

<> Code Pull requests 1 Actions Projects Wiki Security Insights Settings

## add test title #2

Resolving conflicts between `test` and `master` and committing changes → `test`

1 conflicting file

calc.html

```
1 <html>
2 <head>
3
4   <title>Test Calc</title>
5
6 </head>
7 <script language=javascript type="text/javascript">
8   var plus,minus,divide,multiply,max,pow,min,avg
9
```

✓ Resolved

Commit merge

17. Now you can merge the pull request.

## add test title #2

**Open** techupskills wants to merge 1 commit into `master` from `test`

Conversation 0 Commits 1 Checks 0 Files changed 1

techupskills commented 10 minutes ago

No description provided.

add test title 5c6f1ce

Add more commits by pushing to the `test` branch on `techupskills/super_calc`.

Continuous integration has not been set up

GitHub Actions and several other apps can be used to automatically catch bugs and enforce style.

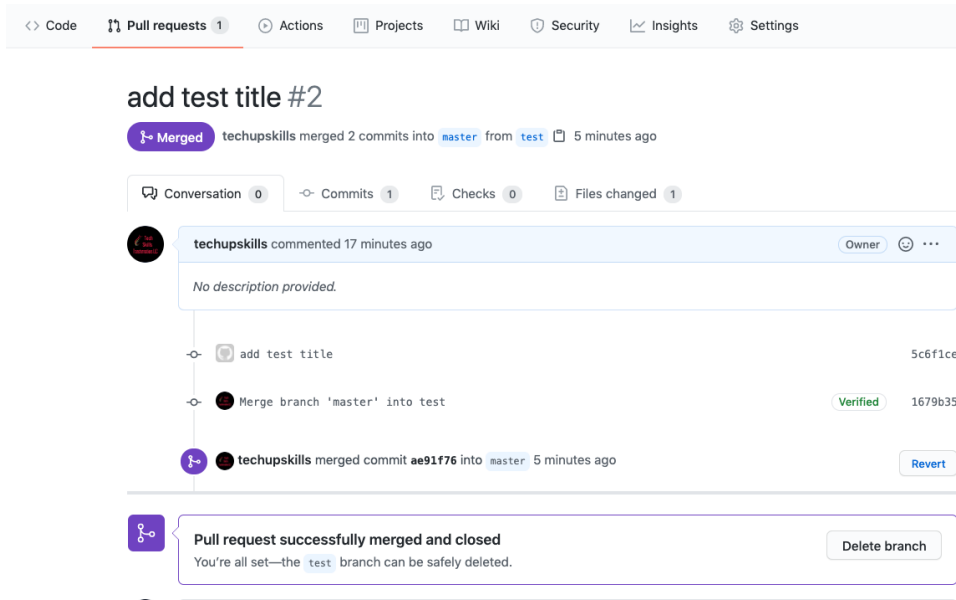
✓ This branch has no conflicts with the base branch

Merging can be performed automatically.

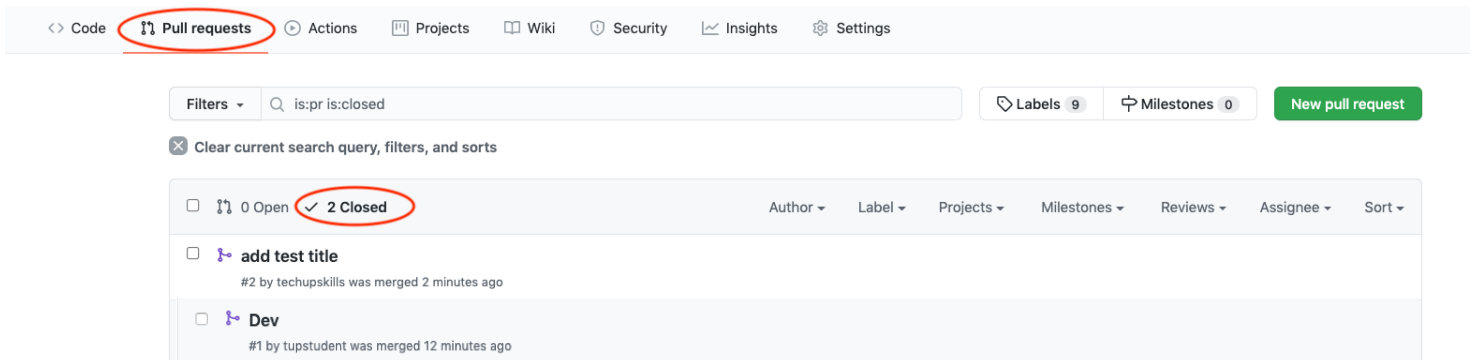
Merge pull request You can also open this in GitHub Desktop or view command line instructions.

Write Preview H B I

18. Click the button to Confirm. And then you'll have another completed pull request.



19. If you click on the "Pull requests" link in the line that begins with "<> Code" you can see the pull request summary. Then if you click on the "2 Closed" link, you can see the ones that you successfully completed.



=====

**END OF LAB**

=====

## Lab 13 - Subtrees

**Purpose:** In this lab, we'll see one way to manage "groups" of repositories in Git

1. Start out in the super\_calc directory for the super\_calc repository that you used in the last two labs. You'll want to be in the master branch.

**\$ git checkout master**

© 2021 Tech Skills Transformations LLC & Brent Laster

2. You're going to add another repository (the sub\_docs one) as a subtree to super\_calc.

```
$ git subtree add -P sub_docs --squash https://github.com/<your github user id>/sub_docs master
```

Even though you don't have much history in this repository, you used the --squash command to compress it. Note that the -P stands for prefix, which is the name your subdirectory gets.

3. Look at the directory structure; note that the sub\_docs subdirectory is there under your super\_calc project. Also, if you do a git log, you can see where the subproject was added and the history squashed.

```
$ ls sub_docs  
$ git log --oneline
```

Note that there is only one set of history here because there is only one project effectively - even though we have added a repository as a subproject.

4. Now, you will see how to update a subproject that is included as a subtree when the remote repository is updated. First, clone the sub\_docs project down into a different area.

```
$ cd ..  
$ git clone https://github.com/<your github user id>/sub_docs sub_docs_temp
```

5. Change into the sub\_docs\_temp project, and create a simple file. Then stage it, commit it, and push it.

```
$ cd sub_docs_temp  
$ echo "readme content" > readme.txt  
$ git add .  
$ git commit -m "Adding readme file"  
$ git push
```

6. Go back to the super\_calc project where you have sub\_docs as a subtree.

```
$ cd ../super_calc
```

7. To simplify future updating of your subproject, add a remote reference for the subtree's remote repository.

```
$ git remote add sub_docs_remote https://github.com/<your github user id>/sub_docs
```

8. You want to update your subtree project from the remote. To do this, you can use the following subtree pull command. Note that it's similar to your add command, but with a few differences:

- You use the long version of the prefix option.

- You are using the remote reference you set up in the previous step.
- You don't have to use the squash option, but you add it as a good general practice.

```
$ git subtree pull --prefix sub_docs sub_docs_remote master --squash
```

Because this creates a merge commit, you will get prompted to enter a commit message in an editor session. You can add your own message or just exit the editor.

9. After the command from step 8 completes, you can see the new README file that you created in your subproject sub\_docs. If you look at the log, you can also see another record for the squash and merge\_commit that occurred.

```
$ ls sub_docs  
$ git log --oneline
```

10. Changes you make locally in the subproject can be promoted the same way using the subtree push command. Change into the subproject, make a simple change to your new README file, and then stage and commit it.

```
$ cd sub_docs  
$ echo "update" >> readme.txt  
$ git commit -am "update readme"
```

11. Change back to the directory of the super\_project. Then use the subtree push command below to push back to the project's remote repository.

```
$ cd ..  
$ git subtree push --prefix sub_docs sub_docs_remote master
```

Note the similarity between the form of the subtree push command and the other subtree commands you've used.

```
=====
```

**END OF LAB**

```
=====
```

#### **Lab 14: Creating a post-commit hook**

**Purpose:** In this lab, we'll see how to put a post-commit hook in place to be active in our local Git environment. The hook could be created in many different programming languages, but we'll use shell scripting here because it's portable among most unix implementations (including the Git bash shell for Windows).

#### **Setup:**

Suppose that you want to mirror out copies of files when you commit them into your local repository - but only for branches that start with "web". Further you must have the config value of hooks.webdir set to a valid directory on your system.

1. The code for the hook is already done for you. Clone down a copy of the repo containing it.

```
$ cd ..
```

```
$ git clone https://github.com/skillrepos/git4-hooks
```

2. In the cloned directory are two files - post-commit-v1.txt and post-commit-v2.txt. You can start out doing this lab with the v1 version, but on some systems you may need to switch out the v2 version if v1 doesn't work as expected. You can also see the current set of sample hooks in the .git/hooks directory.

```
$ cd git4-hooks
```

```
$ ls .git/hooks
```

3. Move (or copy) the post-commit-v1.txt file to the .git/hooks directory as post-commit WITHOUT the extension, and make it executable.

```
$ mv post-commit-v1.txt .git/hooks/post-commit
```

```
$ chmod +x .git/hooks/post-commit
```

4. Create a directory (somewhere outside of your local Git working directory for the hook to eventually mirror your code into.

```
$ mkdir (some-directory-name) (such as mkdir ../mirror)
```

(in your working directory with the git project)

5. In your working directory configure the hooks.webdir value to point to the location you set above.

```
$ git config hooks.webdir (some-directory-name)
```

(make sure to use the correct absolute or relative path to get to the directory you created above, such as `git config hooks.webdir ../mirror`)

6. Next, we'll test out our hook. In your working directory, create a new file, then stage and commit it.

```
$ echo stuff > some-file-name
```

```
$ git add .
```

```
$ git commit -m "new file"
```

7. Notice that after you do the commit, you should see the “Running post-commit hook” message. However, the hook won’t do anything else because the branch is main and not web\*. To verify that’s the case, you can inspect the directory you created for the mirror.

```
$ ls (some-directory-name)
```

There should be nothing there.

8. Now, let’s set things up so our hook will mirror out the contents of our repository. Create a new branch and switch to it. You can name it anything you want as long as it starts with “web”. An example is shown below.

```
$ git checkout -b webtest
```

9. Create another file and stage and commit it into your repository.

```
$ echo stuff > another-file-name
```

```
$ git add .
```

```
$ git commit -m "new file"
```

10. This time, the hook should fire since we have the config value defined and are in a branch that starts with “web”. You should see the “Running” message from the hook and then be able to see the contents of your repository in the directory that you configured.

```
$ ls some-directory-name
```

11. If the hook doesn't seem to work, you may need to repeat step 3 with the post-commit-v2.txt version of the hook and then steps 8 and 9.

```
=====
END OF LAB
=====
```

